
AIMMS Function Reference - System Functions

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com

System Setting Functions

AIMMS supports the following system setting functions, which give access to, or allow modification of, various system settings:

- EnvironmentGetString
- EnvironmentSetString
- OptionGetKeywords
- OptionGetString
- OptionGetValue
- OptionSetString
- OptionSetValue
- ProjectDeveloperMode
- SecurityGetGroups
- SecurityGetUsers
- SolverGetControl
- SolverReleaseControl
- UserColorAdd
- UserColorDelete
- UserColorGetRGB
- UserColorModify

EnvironmentGetString

With the procedure `EnvironmentGetString` you can obtain the string representation of an environment setting.

```
EnvironmentGetString(  
    Key,          ! (input) scalar string expression  
    Value        ! (output) scalar string parameter  
)
```

Arguments:

Key

A string expression holding the name of the environment variable.

Value

A scalar string parameter that, on return, contains the string representation of the current value of the environment variable.

Return value:

The procedure returns 1 if the environment variable exists, or 0 if the name refers to a non-existent environment variable.

Remarks:

- With `EnvironmentGetString` you can retrieve the value for the Windows environment variables like `COMPUTERNAME`, `OS`, `PATH`, `TEMP`, `TMP`, and `USERNAME` as well as environment variables defined by AIMMS like `AIMMSROOT`, `AIMMSBIN`, `AIMSSOLVERS`, `AIMMSCFG`, `AIMMSHELP`, `AIMMSDOC`, `AIMMSUSERDLL`, `AIMMSLOG`, `AIMMSPROJECT`, `AIMMSMODULES`, and `AIMMSTUTORIAL`.

See also:

[EnvironmentSetString](#).

EnvironmentSetString

With the function `EnvironmentSetString` you can set environment variables.

```
EnvironmentSetString(  
    Key,          ! (input) scalar string expression  
    Value        ! (input) scalar string parameter  
)
```

Arguments:

Key

A string expression holding the name of the environment variable.

Value

A scalar string parameter that contains the string representation of the value of you want to assign to the environment variable.

Return value:

The function returns 1 upon success, or 0 otherwise.

Remarks:

- With `EnvironmentSetString` you can change the value for existing environment variables as well as create new environment variables.
- Note that the function `EnvironmentSetString` will only change the values of variables in the environment associated with the AIMMS process.

See also:

[EnvironmentGetString](#).

OptionGetKeywords

With the procedure `OptionGetKeywords` you can obtain set of string keywords, as displayed in the AIMMS **Options** dialog box, that correspond to the numerical (integer) values of an option.

```
OptionGetKeywords(  
  OptionName,      ! (input) scalar string expression  
  Keywords         ! (output) a 1-dimensional string parameter  
)
```

Arguments:

OptionName

A string expression holding the name of the option.

Keywords

A 1-dimensional string parameter that, on return, contains the keywords corresponding to the set of possible (integer) option values.

Return value:

The procedure returns 1 if the option exists, and 0 if the *OptionName* refers to a non-existent option or if the domain set of the 1-dimensional string parameter is too small.

Remarks:

The domain set of the 1-dimensional parameter passed as the *Keywords* argument must have sufficient elements to hold the string keywords of the (integer) option values from the lower bound up to and including the upper bound.

See also:

[OptionGetValue](#), [OptionGetString](#), [OptionSetString](#).

OptionGetString

With the procedure `OptionGetString` you can obtain the string representation of the current value of an AIMMS option, as displayed in the AIMMS **Options** dialog box.

```
OptionGetString(  
  OptionName,          ! (input) scalar string expression  
  CurrentString       ! (output) scalar string parameter  
)
```

Arguments:

OptionName

A string expression holding the name of the option.

CurrentString

A scalar string parameter that, on return, contains the string representation of the current value of the option.

Return value:

The procedure returns 1 if the option exists, or 0 if the name refers to a non-existent option.

Remarks:

Options for which strings are displayed in the AIMMS **Options** dialog box, are represented by numerical (integer) values internally. To obtain the numerical option value, or to obtain the mapping between numerical option values and the corresponding string keywords, you can use the procedures `OptionGetValue` and `OptionGetKeywords`.

See also:

`OptionGetValue`, `OptionGetKeywords`, `OptionSetString`.

OptionGetValue

With the procedure `OptionGetValue` you can obtain the current value of an AIMMS option, as well as its lower and upper bound and default value.

```
OptionGetValue(
  OptionName,      ! (input) scalar string expression
  Lower,           ! (output) scalar numerical parameter
  Current,         ! (output) scalar numerical parameter
  Default,         ! (output) scalar numerical parameter
  Upper           ! (output) scalar numerical parameter
)
```

Arguments:

OptionName

A string expression holding the name of the option.

Lower

A scalar parameter that, on return, contains the lower bound of the possible option values.

current

A scalar parameter that, on return, contains the current (numerical) value of the option.

Default

A scalar parameter that, on return, contains the default (numerical) value of the option.

Upper

A scalar parameter that, on return, contains the upper bound of the possible option values.

Return value:

The procedure returns 1 if the option exists, or 0 if the name refers to a non-existent option or to an option that does not take a number as value.

Remarks:

- Options for which strings are displayed in the AIMMS **Options** dialog box, are also represented by numerical (integer) values. To obtain the corresponding option keywords, you can use the procedures `OptionGetString` and `OptionGetKeywords`.
- You can modify option values programmatically using the `OPTION` statement (see also Section 8.5 of the Language Reference), or using the procedures `OptionSetValue` and `OptionSetString`.

See also:

`OptionGetString`, `OptionGetKeywords`, `OptionSetValue`, `OptionSetString`.

OptionSetString

With the procedure `OptionSetString` you can set the value of a string-valued AIMMS option. You must use the values as displayed in the AIMMS **Options** dialog box.

```
OptionSetString(  
  OptionName,    ! (input) scalar string expressionN  
  NewString      ! (input) scalar string expression  
)
```

Arguments:

OptionName

A string expression holding the name of the option.

NewString

A scalar string expression representing the string representation of the value to be assigned to the option.

Return value:

The procedure returns 1 if the value can be assigned to the option, or 0 if the name refers to a non-existent option, or the value to a non-existent option value.

Remarks:

Options for which strings are displayed in the AIMMS **Options** dialog box, are represented by numerical (integer) values internally. To obtain the numerical option value, or to obtain the mapping between numerical option values and the corresponding string keywords, you can use the procedures `OptionGetValue` and `OptionGetKeywords`.

See also:

`OptionSetValue`, `OptionGetValue`, `OptionGetKeywords`.

OptionSetValue

With the procedure `OptionSetValue` you can set the value of a numeric AIMMS option. The value assigned to the option must be contained in the option range displayed in the AIMMS **Options** dialog box.

```
OptionSetValue(  
    OptionName,      ! (input) scalar string expression  
    NewValue         ! (input) scalar numeric expression  
)
```

Arguments:

OptionName

A string expression holding the name of the option.

NewValue

A scalar numeric expression representing the new value to be assigned to the option.

Return value:

The procedure returns 1 if the option exists and the value can be assigned to the option, or 0 otherwise.

Remarks:

- Options for which strings are displayed in the AIMMS **Options** dialog box, are also represented by numerical (integer) values. To obtain the corresponding option keywords, you can use the procedures `OptionGetString` and `OptionGetKeywords`.
- You can also modify option values using the `OPTION` statement (see also Section 8.5 of the Language Reference).

See also:

`OptionGetString`, `OptionGetKeywords`, `OptionSetString`.

ProjectDeveloperMode

The function `ProjectDeveloperMode` indicates whether a project is opened in developer or end-user mode.

`ProjectDeveloperMode`

Arguments:

None

Return value:

The function returns 1 if the project is opened in developer mode, or 0 if the project is opened in end-user mode.

SecurityGetGroups

With the procedure `SecurityGetGroups` you can fill a set with group names from the user database that is linked to the project.

```
SecurityGetGroups(  
    group_set      ! (output) an (empty) root set  
)
```

Arguments:

group_set

A root set, that on return will contain elements that represent all group names from the user database.

Return value:

The procedure returns 1 on success, and 0 on failure.

See also:

The procedure [SecurityGetUsers](#).

SecurityGetUsers

With the procedure `SecurityGetUsers` you can fill a set with user names from the user database that is linked to the project. You can filter which users are included in the set based upon their group or authorization level.

```
SecurityGetUsers(  
  user_set,      ! (output) an (empty) root set  
  [group,]      ! (optional) scalar string  
  [level]       ! (optional) element of the set AllAuthorizationLevels  
)
```

Arguments:

user_set

A root set, that on return will contain elements that represent the user names from the user database.

group (optional)

A string representing a group name from the user database. If specified, then only the users that belong to this group are returned.

level (optional)

An element of the set `AllAuthorizationLevels`. If specified, then only the users that have the specified authorization level are returned.

Return value:

The procedure returns 1 on success, and 0 on failure.

See also:

The procedure [SecurityGetGroups](#).

SolverGetControl

A single use local license allows you to run two concurrent AIMMS sessions. At any time, however, only one of these sessions can make use of a solver. Prior to executing a SOLVE statement, AIMMS will determine whether the solver is already locked by another session. If this is the case, AIMMS will abort the SOLVE statement with a runtime error. If the solver is not locked, AIMMS locks the solver for the duration of SOLVE statement by default. With the procedure `SolverGetControl` you can programmatically lock the solver for a prolonged period of time, for instance, during an algorithm requiring multiple solves.

`SolverGetControl`

Arguments:

None

Return value:

The procedure returns 1 if the solver was successfully locked, or 0 otherwise.

Remarks:

- AIMMS also supports multi-session local licenses that allow you to run multiple concurrent solves, and twice that number of concurrent AIMMS sessions.
- This procedure has no effect if you are connecting to an AIMMS network license server. In that case every session requires a separate floating network license.

See also:

The procedure `SolverReleaseControl`.

SolverReleaseControl

A single use local license allows you to run two concurrent AIMMS sessions. At any time, however, only one of these sessions can make use of a solver. Prior to executing a SOLVE statement, AIMMS will determine whether the solver is already locked by another session. If this is the case, AIMMS will abort the SOLVE statement with a runtime error. If the solver is not locked, AIMMS locks the solver for the duration of SOLVE statement by default. With the procedure `SolverReleaseControl` you can unlock a solver previously locked by a call to the procedure `SolverGetControl`.

`SolverReleaseControl`

Arguments:

None

Return value:

The procedure returns 1 if successful, or 0 if the solver was not currently locked by this session.

Remarks:

- AIMMS also supports multi-session local licenses that allow you to run multiple concurrent solves, and twice that number of concurrent AIMMS sessions.
- This procedure has no effect if you are connecting to an AIMMS network license server. In that case every session requires a separate floating network license.

See also:

The procedure `SolverGetControl`.

UserColorAdd

With the procedure `UserColorAdd` you can programmatically add a new color to the set of user colors.

```
UserColorAdd(  
  color_name,      ! (input) scalar string expression  
  red,             ! (input) scalar numerical expression  
  green,          ! (input) scalar numerical expression  
  blue            ! (input) scalar numerical expression  
)
```

Arguments:

color_name

A string expression holding the name of the user color to add.

red

An integer value in the range 0...255 indicating the red component in the RGB value of the color.

green

An integer value in the range 0...255 indicating the green component in the RGB value of the color.

blue

An integer value in the range 0...255 indicating the blue component in the RGB value of the color.

Return value:

The procedure returns 1 if the color could be added successfully, or 0 if the color already exists.

Remarks:

Only project colors, i.e. colors added through the **Tools-User Colors** dialog box, are persistent. User colors that are added to a project using the procedure `UserColorAdd` do not persist, and, therefore, have to be added during the initialization of every project session.

See also:

[UserColorDelete](#), [UserColorGetRGB](#), [UserColorModify](#). User colors are discussed in full detail in Section 11.4 of the User's Guide.

UserColorDelete

With the procedure `UserColorDelete` you can programmatically delete a color from the set of user colors.

```
UserColorDelete(  
    color_name      ! (input) scalar string expression  
)
```

Arguments:

color_name

A string expression holding the name of the user color to delete.

Return value:

The procedure returns 1 if the color could be deleted successfully, or 0 if the color does not exist, or is contained in the fixed set of project colors.

Remarks:

You can only delete user colors that have been added using the procedure `UserColorAdd`. Colors added through the **Tools-User Colors** dialog box are fixed and cannot be deleted or modified.

See also:

[UserColorAdd](#), [UserColorGetRGB](#), [UserColorModify](#). User colors are discussed in full detail in Section 11.4 of the User's Guide.

UserColorGetRGB

With the procedure `UserColorGetRGB` you can programmatically obtain the RGB values of a color in the set of user colors.

```
UserColorGetRGB(  
    color_name,      ! (input) scalar string expression  
    red,             ! (output) scalar numerical parameter  
    green,           ! (output) scalar numerical parameter  
    blue             ! (output) scalar numerical parameter  
)
```

Arguments:

color_name

A string expression holding the name of the user color to query.

red

An scalar parameter that, on return, holds the red component in the RGB value of the color.

green

An scalar parameter that, on return, holds the green component in the RGB value of the color.

blue

An scalar parameter that, on return, holds the blue component in the RGB value of the color.

Return value:

The procedure returns 1 if the color exists in the set of user colors, or 0 if the color does not exist.

See also:

`UserColorAdd`, `UserColorDelete`, `UserColorModify`. User colors are discussed in full detail in Section 11.4 of the User's Guide.

UserColorModify

With the procedure `UserColorModify` you can programmatically modify an existing color in the set of user colors.

```
UserColorModify(  
    color_name,      ! (input) scalar string expression  
    red,             ! (input) scalar numerical expression  
    green,          ! (input) scalar numerical expression  
    blue            ! (input) scalar numerical expression  
)
```

Arguments:

color_name

A string expression holding the name of the user color to modify.

red

An integer value in the range 0...255 indicating the red component in the RGB value of the color.

green

An integer value in the range 0...255 indicating the green component in the RGB value of the color.

blue

An integer value in the range 0...255 indicating the blue component in the RGB value of the color.

Return value:

The procedure returns 1 if the color could be modified successfully, and 0 if the color does not exist, or is contained in the fixed set of project colors.

Remarks:

You can only modify user colors that have been added using the procedure `UserColorAdd`. Colors added through the **Tools-User Colors** dialog box are fixed and cannot be deleted or modified.

See also:

`UserColorAdd`, `UserColorDelete`, `UserColorGetRGB`. User colors are discussed in full detail in Section 11.4 of the User's Guide.